

**Code Review 1**

**Data Structures and Algorithms**  
**for AMR Applications in Titanium**

Tong Wen

Phil Colella

LBL

April 8, 2003

## Overview

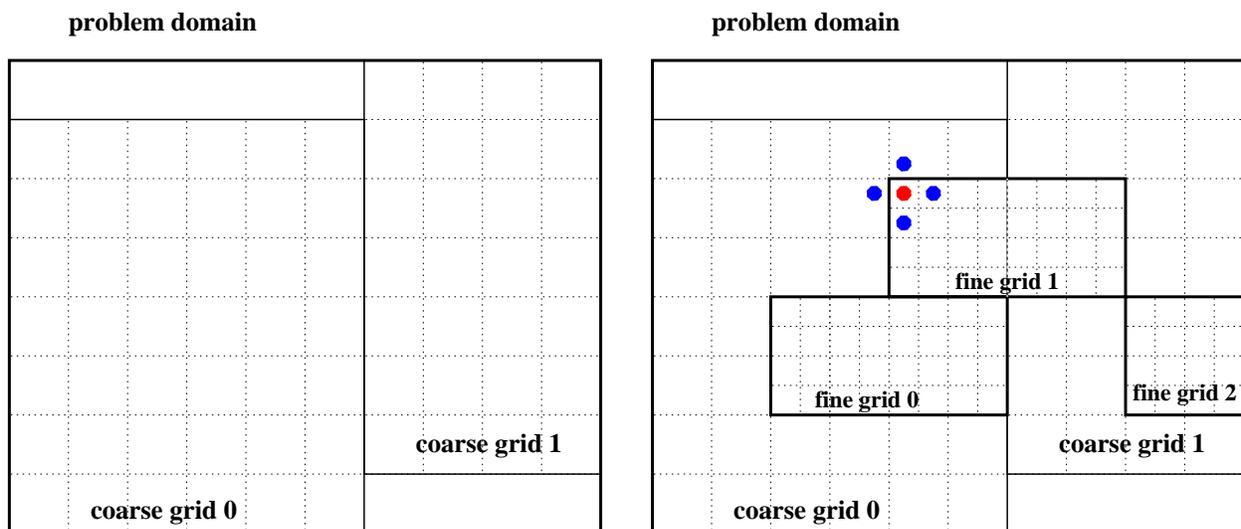
---

1. Introduction to Adaptive Mesh Refinement (AMR).
2. What we have in packages BoxTools and AMRTools.
3. AMR data structures: review and update on BoxTools.
  - BoxLayout: the metadata.
  - BoxLayoutData: the distributed-data container.
  - Timing results.
4. AMR algorithms: an example.
  - Quadratic coarse-fine boundary interpolation.
  - AMRTools.QuadCFInterp: our implementation.

# Adaptive Mesh Refinement

---

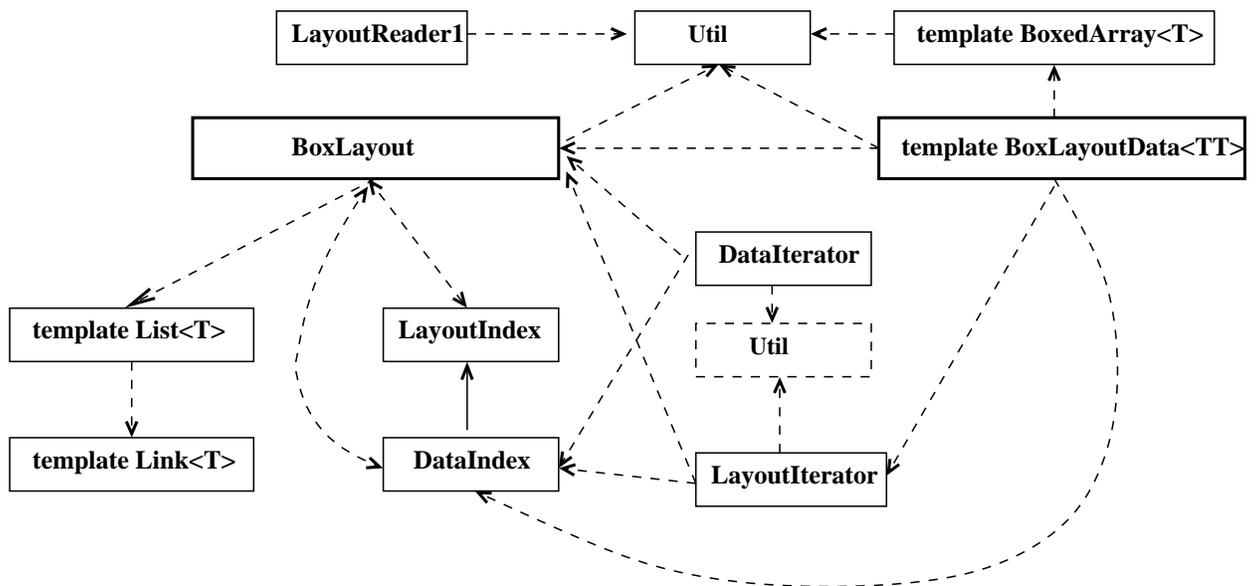
1. Motivation.
2. No free lunch: problems caused by local refinement.
3. Our goal.



# An overview of BoxTools package

---

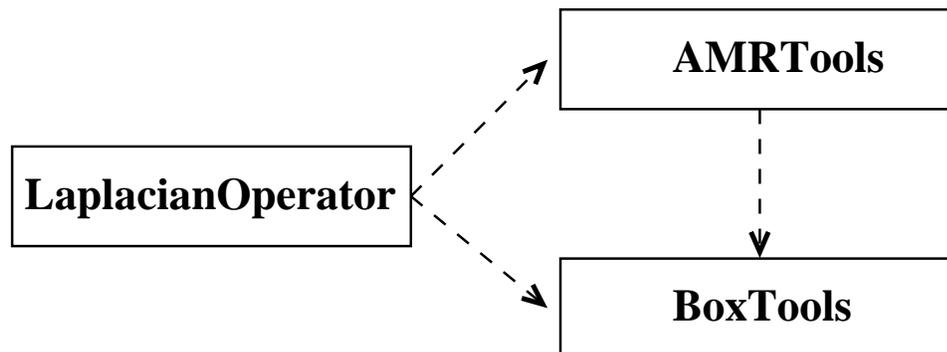
- BoxTools contains classes for basic data structures:
  - BoxLayout, LayoutIterator, LayoutIndex;
  - BoxedArray;
  - BoxLayoutData, DataIterator, DataIndex;
  - Util, LayoutReader1;
  - List, Link.
- Collaboration diagram:



## An overview of AMRTools package

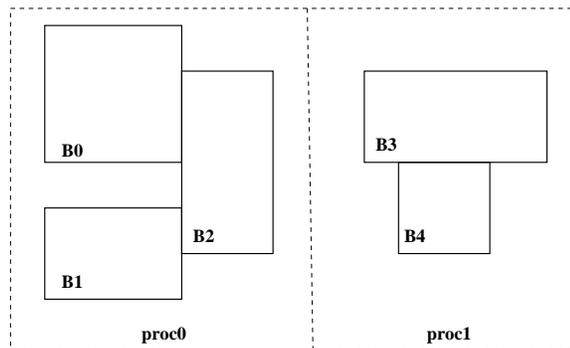
---

- AMRTools contains classes for basic AMR algorithms:
  - QuadCFInterp;
  - CoarseAverage, FineInterp, ..., LevelFluxRegister.
- Collaboration diagram:

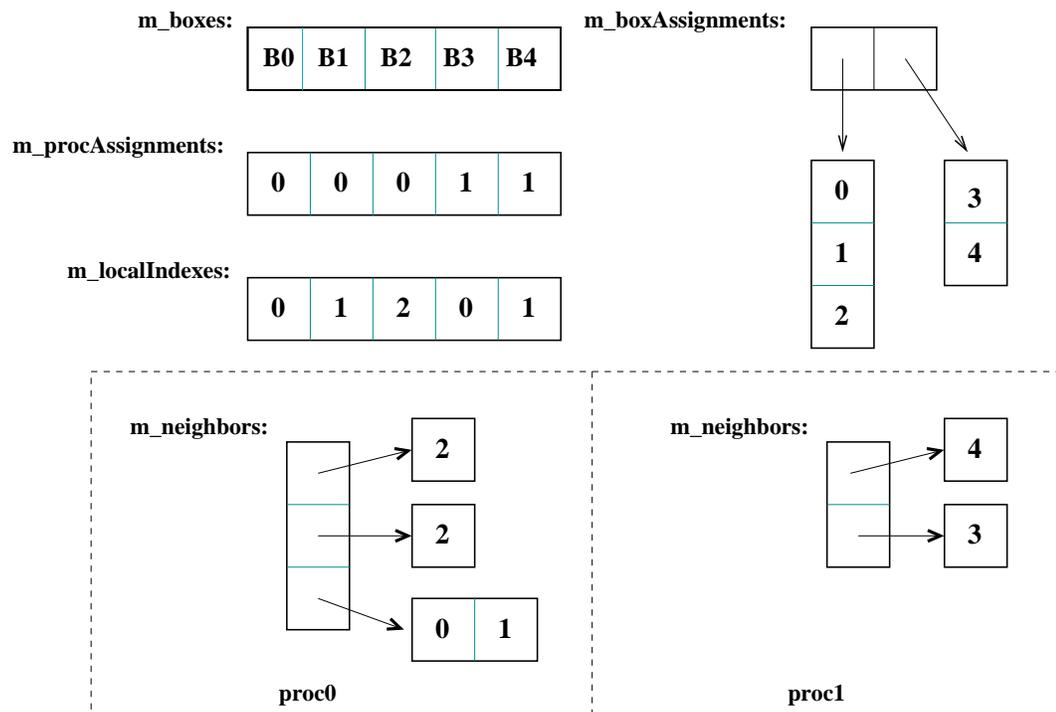


# BoxLayout class

- A layout of Boxes (RectDomains) at the same refinement level:



- Data structures used to represent the layout information:



## BoxLayout: key fields

---

- Key fields:
  - m\_boxes:  
`protected RectDomain<SPACE_DIM> [1d] local m_boxes;`
  - m\_procAssignments:  
`protected int [1d] local m_procAssignments;`
  - m\_localIndexes:  
`protected int [1d] local m_localIndexes;`
  - m\_boxAssignments:  
`protected int [1d] local [1d] local m_boxAssignments;`
  - m\_neighbors:  
`protected int [] local [] local m_neighbors;`
- Note that
  - allocating and deallocating temporary data structures (Lists) are controlled by Regions.
  - 1D Ti arrays vs 1D Java arrays.

## BoxLayout: key methods and operators

---

- **constructor:**

```
public BoxLayout(RectDomain<SPACE_DIM> single []  
single a_Boxes, int single [] single a_procIDs)
```

- **define:**

```
public final local void define(RectDomain<SPACE_DIM>  
single [] single a_Boxes, int single [] single a_procIDs)
```

- **op[ ]:**

```
public final inline local RectDomain<SPACE_DIM> op[](  
LayoutIndex local a_index)
```

- **copy:**

```
final local BoxLayout local single copy()
```

- **refine & coarsen:**

```
public final local BoxLayout local single refine(  
int single a_factor)  
public final local BoxLayout local single coarsen(  
int single a_factor)
```

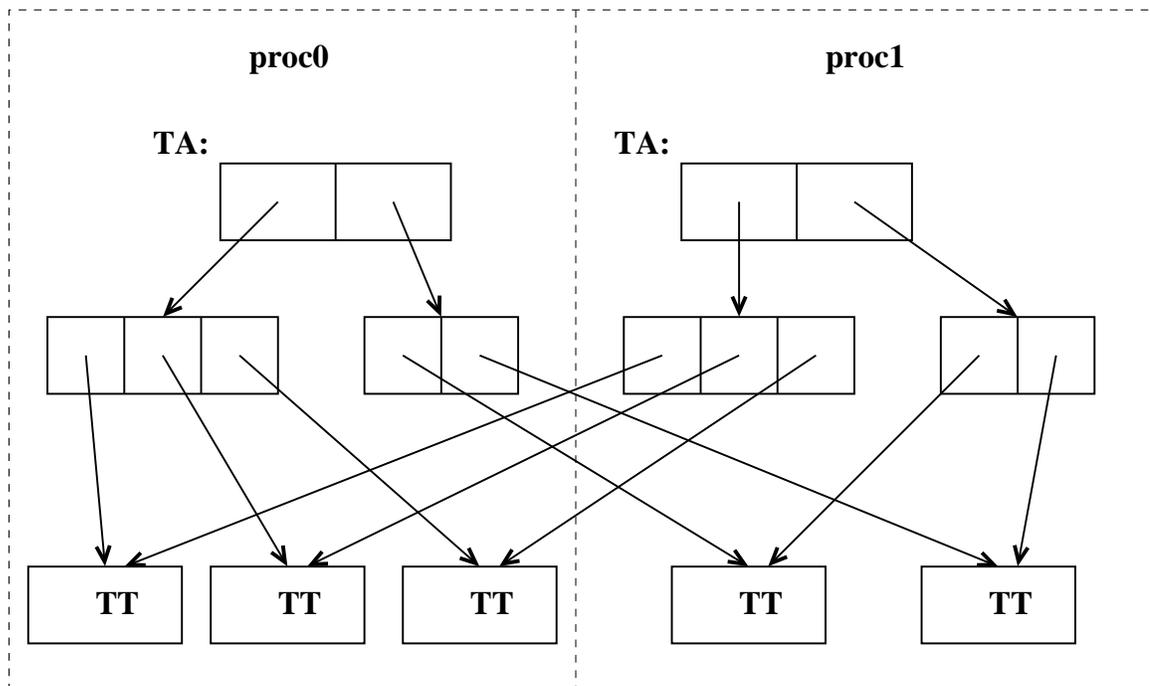
- **lowBoundary & highBoundary:**

```
public final local BoxLayout local single lowBoundary(  
int single a_direction, int single a_length,  
boolean single a_expendBy1)  
public final local BoxLayout local single highBound-  
ary(int single a_direction, int single a_length,  
boolean single a_expendBy1)
```

## BoxLayoutData class

---

- BoxLayoutData is a template class with TT as the template parameter.
- The key field TA:  
`protected TT [1d] local [1d] local TA;`
- TT is a box-oriented data type, such as `template BoxedArray<T>`.
- exchange vs broadcast.



## BoxLayoutData: key methods and operators

---

- **constructor:**

```
public inline BoxLayoutData(BoxLayout local single a_layout,  
int single a_ghostSize)
```

- **define:**

```
public final local single void define(  
BoxLayout local single a_layout, int single a_ghostSize)
```

- **op[ ]:**

```
public final inline local TT local op[ ](  
DataIndex local a_DI)
```

- **exchange:**

```
public final local single void exchange()
```

- **copy:**

```
public final local single void copy(  
template BoxLayoutData<TT > local single BLD_src,  
boolean single ghostIncluded)
```

## BoxTools: timing results

---

- The £nest level contains 1857 boxes(RectDomains) and totally 15344640 points.
- Timings on `ford.lbl.gov` (in milliseconds):

Operations	-bcheck	-nobcheck	-nobcheck localCopy() copy()
Reading the data £le	395	361	395
Constructing the BoxLayout	370	361	355
Coarsening the BoxLayout	4	4	4
Finding neighbors ♡	943	787	791
Constructing one BoxLayoutData ♠	654	664	659
Initialing one BoxLayoutData ♡	603	♣335	341
Doing the exchange ♡	1058	935	♣353
One sweep of Laplacian operation ♡	942	828	800
Copying BoxLayoutData ♡	1826	1677	♣653

## BoxTools: timing results

---

- Timings on seaborg.nersc.gov (# of procs=1):

Operations	old	new
Reading the data £le	2189	2253
Constructing the BoxLayout	639	541
Coarsening the BoxLayout	12	♣35
Finding neighbors ♡	4246	♣2799
Constructing one BoxLayoutData ♠	1162	1101
Initialing one BoxLayoutData ♡	542	536
Doing the exchange ♡	2183	♣586
One sweep of Laplacian operation ♡	633	622
Copying BoxLayoutData ♡	5371	♣593

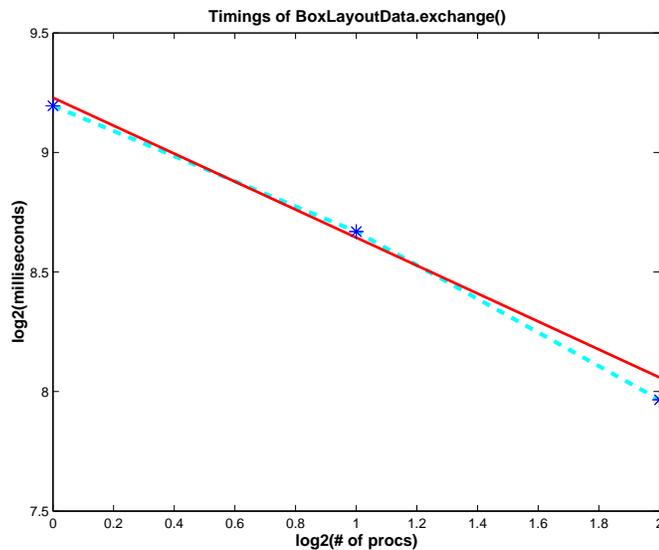
## BoxTools: timing results

---

- Timings on seaborg.nersc.gov (# of procs=1, 2, 4):

Operations	# of procs		
	1	2	4
Reading the data £le	2253	2563	2419
Constructing the BoxLayout	541	563	601
Coarsening the BoxLayout	35	9	10
Finding neighbors ♡	2799	2151	1071
Constructing one BoxLayoutData ♠	1101	1285	1385
Initialing one BoxLayoutData ♡	536	275	142
Doing the exchange ♡	586	407	250
One sweep of Laplacian operation ♡	622	319	169
Copying BoxLayoutData ♡	593	302	156

- Scalability:



## BoxedArray.localCopy() vs BoxedArray.copy()

---

- copy():

```
public final inline local void copy(  
    template BoxedArray<T> from_BA,  
    RectDomain<SPACE_DIM> from_domain)  
{  
    m_array.copy(from_BA.getArray().restrict(from_domain));  
}
```

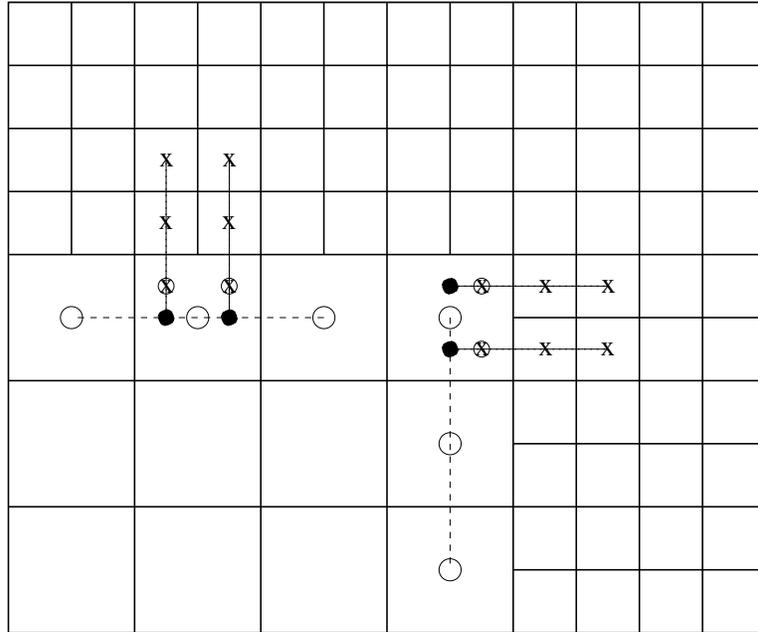
- localCopy():

```
public final inline local void localCopy(  
    template BoxedArray<T> local from_BA,  
    RectDomain<SPACE_DIM> from_domain)  
{  
    T [SPACE_DIM d] local tiArrayB=from_BA.getLocalArray();  
    RectDomain<SPACE_DIM> box=from_domain*m_domain;  
    Point<SPACE_DIM> point;  
    foreach (point in box) m_array[point]= tiArrayB[point];  
}
```

- localCopy() vs copy():

Operations	# of procs			
	ford	1	2	4
Doing the exchange (not using copy())	353	586	407	250
			408	250
Copying BoxLayoutData (using copy())	653	593	302	156
	655	700	1974	939

## Quadratic coarse-fine boundary interpolation



1. Given  $i$ , there is a unique choice of  $\pm$  and  $d$ , such that  $i \mp e^d \in \Omega_k^f$ .
2. Interpolation in the direction orthogonal to  $d$ :

$$\tilde{\varphi}_{i \pm \frac{1}{2}e^d} = \varphi_{i^c} + \sum_{d' \neq d} \left[ (x_{d'} (D^{1,d'} \varphi^c)_{i^c} + \frac{1}{2} (x_{d'})^2 (D^{2,d'} \varphi^c)_{i^c}) + \sum_{d'' \neq d, d'' \neq d'} x_{d'} x_{d''} (D^{d'd''} \varphi^c)_{i^c} \right]$$

where

$$x = \frac{i + \frac{1}{2}u}{n_{ref}} - (i^c + \frac{1}{2}u).$$

3. Interpolation in the normal direction:

$$\tilde{\varphi}_i = 4a + 2b + c$$

where

$$a = \frac{\tilde{\varphi}_{i \pm \frac{1}{2}e^d} - (n_{ref} \cdot |x_d| + 2)\varphi_{i \mp e^d} + (n_{ref} \cdot |x_d| + 1)\varphi_{i \mp 2e^d}}{(n_{ref} \cdot |x_d| + 2)(n_{ref} \cdot |x_d| + 1)}$$

$$b = \varphi_{i \mp e^d} - \varphi_{i \mp 2e^d} - a$$

$$c = \varphi_{i \mp 2e^d}$$

## QuadCFinterp class

---

- coarseFineInterp method:

```
public final local void coarseFineInterp(template
BoxLayoutData<template BoxedArray<T>> local single
a_phif, template BoxLayoutData<template BoxedArray<T>>
local single a_phic)
```

- implementation:

```
        . . .
/**** Loop 1: over the orientations ****/
for (orient=2;orient<2*(SPACE_DIM+1);orient++){
    . . .
    LI.reset();
    /** Loop2: over the layout at the fine level **/
    while (!LI.isEnded()){
        . . .
        /** Loop3: over each Ti array **/
        foreach (p in boxCut){
            . . .
            interpolation in the direction
                orthogonal to current orientation;
            interpolation in the current orientation;
            . . .
        }
        . . .
        LI.advance();
    }
}
```

## What is the next?

---

- Language supports?
  1. adding the length field to 1D Ti arrays?  
`for (int i=0;i<Ti_array.length;i++) { ... }`
  2. adding the method to judge whether two RectDomains overlap?
- implementing AMRTools.LevelFluxRegister.
- implementing the Laplacian operator.